School of Electronic Engineering
and Computer Science

Final Report

Programme of study:
BSc Computer Science

Project Title:
Exploring Automatic Accented Musical Onset Detection

Supervisor:
Johan Pauwels

Student Name:
Kazeo Kazimierz Hada

Final Year
Undergraduate Project 2023/24

Queen Mary
University of London

Date: 28th April 2024

Abstract

Automatic Music Transcription is a highly important field of Music Information Research. It involves detecting the notes played in a piece of music, their duration, and the frequencies they are played in. The goal of Automatic Transcription is to effectively translate a musical audio input into sheet music. While a state-of-the-art model may produce satisfying outputs, it still omits important details about the music. Those could include features such as articulations, dynamics, and expressions, which are essential in giving character to a piece of music.

This project aims to extend an Onset Detection model, one of the components of an Automatic Music Transcription program, to enable the detection of notes played with the "accent" articulation. Since no known research has been made on this topic, there is no existing open-source dataset for training such a model. Therefore, the project also involved writing a data pipeline for programmatically generating such a dataset.

The generated dataset contained 312 different snippets, all with different instruments, rhythms, and randomised instances of accents on notes. An open-source Onset Detection Convolutional Neural Network model has been modified to suit the multilabel classification task that is Accented Onset Detection. The model has been trained using 8-fold cross-validation for 20 epochs.

The results of the model's predictions were mixed. While it still performed very well in detecting onsets despite the modifications, it underperformed in identifying accented notes. This may be due to the imbalance present in the dataset, or the structure of the model itself. Nevertheless, we hope that this work could inspire other MIR researchers to study Accented Onset Detection and outdo the results observed in this project.

Keywords: Accent, Onset, Onset Detection, Dataset, Convolutional Neural Network, Multilabel

Acknowledgements

I would like to thank Dr. Johan Pauwels for his incredible support and interest in this project, as well as Sebastian Böck for courteously granting me access to his Böck dataset.

I would also like to thank my family and friends for supporting me through the highs and lows of my full academic journey at Queen Mary University of London.

Contents

Chapter 1: **Introduction**

1.1 Background

Music Information Retrieval (MIR) is an interdisciplinary research field concerned with the extraction of features and characteristics from music and analysing or categorising them (Pocaro n.d.).

Early MIR research focused on working with digital representations of music pieces such as MIDI, but the emergence of signal processing technology as a result of increased computing power enabled the broadening of MIR research to analyse musical recordings (Schedl et al 2014). Today, MIR technologies are used in various real-world applications such as music recommender systems, pitch tracking, beat tracking, key detection, genre recognition, cover song detection or audio fingerprinting (Schedl et al 2014).

One of the most complex branches of MIR is Automatic Music Transcription (AMT), as it requires three other branches of MIR to work together; pitch detection, onset detection and possibly instrument detection, in order to achieve a functional AMT system.

Onset detection plays a pivotal role in AMT, as studies show it is more difficult to recognise the timbre of notes with removed onsets. The most simple detection algorithms have been written for beat and rhythm tracking systems, which aren't accurate enough to be used in AMT systems, as they only respond to very prominent onsets in a signal (Marolt et al 2002). The task of onset detection becomes more challenging when faced with polyphonic music audio, as opposed to monophonic audio, where there is clear separation between onsets. Polyphonic signals contain different notes playing at the same time, creating a fuzzy signal which reveals very little about what is going on in individual frequency regions of the signal (Marolt et al 2002).

Apart from AMT, onset detection systems are also useful for tempo extraction, music editing, music classification and music fingerprinting (Lacoste and Eck 2005).

1.2 Problem Statement

In music notation, articulations indicate the "character" of the note to be played. The most common accents are the standard accent, marcato, staccato, and tenuto, displayed in Figure 1. The standard accent, represented by a wedge facing to the right, represents a note played using a strong attack and softening shortly after. A note with a marcato, represented by a vertical wedge, is played similarly to an accented note, using a strong attack but with an early release. The staccato mark, represented by a dot, indicates that a note should be played short, generally half the duration of the note value. It is used to create separation between that note and the next note. Finally, the tenuto accent indicates that the note should be held at its full length. Those articulations can also be stacked on top of each other, i.e. an accent over a tenuto, resulting in a note with a strong attack, and held at its full length.

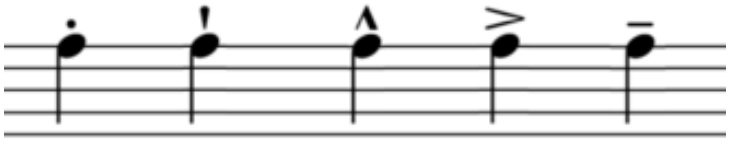


Figure 1. Notation of articulations in music. From left to right: Staccato, Staccattissimo, Marcato, Accent, Tenuto. Source:

There is no single piece of music that does not make use of articulations. Articulations, among other music notation elements, such as dynamics, suspensions, tempo or ornaments, add character to a melody, distinguish different parts of a musical piece, and play a role in the emotional development of a musical composition. Taking away those elements would result in taking away a lot of personality from the piece of music as well as from the musician playing it.

Currently, AMT models do not take into consideration the articulations of the particular note but only look for instances where notes are present, how long they last, and what the pitch of each particular note is. While that already is a great feat in the MIR field, it still doesn't accurately represent a piece of music.

1.3 Aim

The aim of the project is to extend an existing Onset Detection Model, to make it able to recognise whether the onsets that it detects are accented or not. However, since there is no public dataset available to train such a model, a dataset including musical snippets with instances of accented notes will be created. Due to time and financial constraints, I won't be able to record said snippets on my own. Manually playing and recording snippets would be impossible for the scope of this project due to time and financial constraints. Therefore the dataset will contain snippets of digital music with randomised instances of accents programmatically generated by a data pipeline.

To stay within the scope of a Bachelor project, the project will solely focus on the "accent" articulation when developing the machine learning model. The dataset will also include snippets with polyphonic music, as the most refined AMT models as of today are capable of accurately processing polyphonic music.

1.4 Objectives

The project will be broken down into 4 objectives, to be completed sequentially.

Firstly, a data pipeline will be coded, which will be used to programmatically manipulate digital music snippets and add accents in random instances of the snippet. Those snippets will come from the Music21 Python library, which contains a large corpus of music from different genres. The library allows for modifying the music included in the corpus and exporting the generated pieces in different formats. The data pipeline will create a dataset containing audio files corresponding to the transformed snippets, and records of what onsets include an accent.

Then, with the created data pipeline, a proper dataset for training the machine learning model will be generated. The dataset will be based on datasets that were used

for training existing Onset Detection models, as a preexisting model will be extended as opposed to writing one from the ground up.

Once the dataset is ready, a preexisting open-source Onset Detection Model will be extended by adding the functionality of detecting whether an onset is played with an accent articulation or not. Finally, the model will be trained and its results will be analysed.

1.5 Research Question

Can accents be automatically detected from a musical snippet using a Neural Network?

Chapter 2: **Literature Review**

Most onset detection methodologies are broken down in three steps: Preprocessing, used to emphasise the relevant parts of the signal, the Onset Detection Function (ODF), which is the core function of the Onset Detection model, and where most of the research is focused, and Peak-Picking, where onsets are extracted from the ODF (Eyben et al 2010).

Those steps may be called differently by other researchers, but are all fundamentally tasked to do the same thing. We will use the same nomenclature as above to avoid confusion.

The first Onset Detection algorithms were created using signal processing techniques, with researchers applying psychoacoustic knowledge to mimic human perception of sound in their algorithms. The algorithm developed by Klapuri (1999) is one example of a detection algorithm based on psychoacoustic knowledge. At the algorithm's preprocessing step, the signal is split into distinct frequency bands by a filter bank as that is how the human auditory system treats sound.

Klapuri (1999) found that the method used in previous Onset Detection research, using the first-order difference function and taking the maximum rising slope of an onset fails to precisely mark the time of the onset. That is because low sounds may take time to come to the point where their amplitude is maximally rising, that point being late from the actual onset of the sound, and because the onset track of a sound is not monotonically increasing, which means that several local maxima would appear in the first order difference function. Klapuri (1999) proposed using the relative difference function, i.e. looking at the amount of change in amplitude in relation to the signal level, to detect onset components and determine when they appear. This approach is psychoacoustically relevant, since humans perceive an increase in signal amplitude in relation to its level (Klapuri 1999). At the peak-picking step, the algorithm looks for onset candidates above a global threshold value and drops out those that fall below, and because a single onset is prone to having many closely spaced peaks in each band, all peaks within a 50ms time window are merged together.

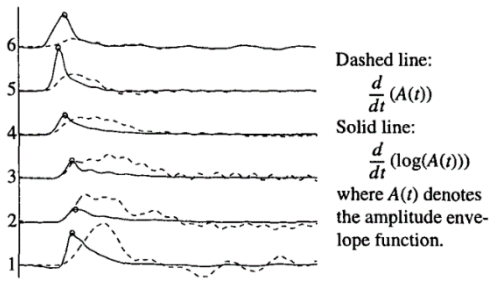


Figure 2. Onset of a piano sound. First order *absolute* (dashed) and *relative* (solid) difference functions of the amplitude envelopes of six different frequency bands.

Figure 2 Illustrates difference between first-order and relative difference function. (Klapuri 1999)

Klapuri (1999) tested their algorithm on 10-second excerpts from performances of different genres of polyphonic music. It was found that onsets for different instruments were well detected by the software, bar the recordings of music played by a symphonic orchestra. Klapuri (1999) supposes that it is because individual physical sound sources couldn't be followed in such large ensembles.

Bello and Sandler (2003) proposed the usage of phase information for developing ODFs, as it carries all the timing information about an audio signal. They defined that a note is composed of two components: the transient region, followed by the steady state.

The transient region is short and contains rapid changes in the signal spectra. The steady-state follows the transient region, it is when the signal is stationary and predictable. They concluded that in order to find the onset of a note, the ODF needs to find at what moment the transient region begins.

To achieve this, Bello and Sandler (2003) wrote a transient/steady-state separation algorithm, in order to separate both components of a note and statistically analyse the results to pinpoint where the onset is. The statistical analysis is conducted by calculating several variables. The standard deviation quantifies the variation of the signal. It characterises onsets as peak/valley pairs but presents a noisy profile, the several local peaks surrounding each global peak could lead to inaccurate results. Therefore, they smoothened the signal profile of the standard deviation by only representing the Interquartile range (IQR) of each signal frame. Lastly, they use Kurtosis to measure the flatness of the signal, which is located where steady states are present in the signal variation.

To find onsets in a signal, Bello and Sandler (2003) first located where peaks occurred in the Kurtosis profile, indicating high flatness in the signal variation, i.e. beginnings of a steady state, and backtracked to find the closest preceding peak in the IQR within a certain range. They claim to have found detection rates between 80% and 90% with a 10% rate of false positives (Bello and Sandler 2003).

With the emergence and success of Neural Networks in other research fields, MIR researchers quickly started to use Neural Networks in their own research.

One of the first examples of an Onset Detection algorithm developed using Neural Networks was by Marolt et al (2002). While working on SONIC, their transcription system for polyphonic piano music, they used a simple Onset Detection algorithm, i.e. making the onset of the note equal to the time of its finding, but found several inaccuracies with it, especially when locating onsets for notes in lower octaves, where the onsets would be determined with significant delays. They have instead opted to develop a separate, more robust Onset Detection system for SONIC. After experimenting with Klapuri's (1999) algorithm, they found that the peak-picking function was very sensitive to the choice of thresholds – lower values would produce many false onsets and higher values would omit a lot of onsets (Marolt et al 2002).

Marolt et al (2002) split the signal into 22 frequency bands, each processed with a different filter. The difference filter equation calculates the difference in variation over time of the signal's amplitude. The Onset Detection algorithm looked at the outputs of the difference filters and determined which peaks corresponded to a note onset, and which to noise in the signal. The Neural Network of the algorithm is composed of a fully connected network of integrate-and-fire neurons and a multilayer perceptron. Upon detecting an onset, a neuron would fire an impulse that would be used as an input to the multilayer perceptron (MLP). The MLP would use this impulse as well as several other factors, to decide whether an onset occurred or not. The model was trained and tested exclusively on piano recordings, both synthesised and real. Marolt et al (2002) claim that the average score of their system reached 98% of correctly found onsets, with better results on synthesised recordings, due to outside factors disturbing the signal of a real piano recording.

Lacoste and Eck (2005) wrote the Onset Detection algorithm that won the first Music Information Retrieval Evaluation eXchange (MIREX) contest in 2005. They have

developed two Feedforward Neural Network (FNN) algorithms, called SINGLE-NET and MANY-NETS, which both attained the two highest F-scores of the contest on the MIREX dataset. (Lacoste and Eck 2005) Initially, they aimed to transform the input signal with a time-frequency transform, either the Short-Time Fourier Transform (STFT) or the Constant-Q Transform (CQT), and Bello and Sandler's (2003) phase plane transform, combine their results in the FNN, in order to make use of both magnitude plane based and phase plane based methodologies. However, they found that the addition of the phase plane did not yield better results, and seemed only to be helpful to find trivial onsets. They therefore ended up combining results of the linear and logarithmic frequencies of the signal's spectrogram, which can be both found using STFT and CQT.

The SINGLE-NET algorithm followed the standard three-step process that most Onset Detection algorithms used. In order to address ambiguities with certain detected onsets, Lacoste and Eck (2005) decided to use a tempo-based approach to determine whether peaks in the signal are onsets or not based on whether or not they rhythmically fit with the rest of the onsets. This approach is used in the MANY-NETS algorithm. In the MANY-NETS algorithm, the SINGLE-NET algorithm is first repeated 5 times, each time with different parameters, such as the number of input variables, window width, and frequency range, so that each iteration produces complementary, but not identical results. While the SINGLE-NET algorithm loops, the tempo is traced based on the onsets received by the multiple iterations. The output then results in a map of the instants of onsets and the probability that the onset matches the rhythmic structure of the song. This output is fed to a final SINGLE-NET network, which is tasked to make the decision whether or not each peak is an onset based on its probability. The SINGLE-NET and MANY-NETS algorithms achieved an F-score of 78.35% and 80.07% respectively on the MIREX dataset. (Lacoste and Eck 2005).

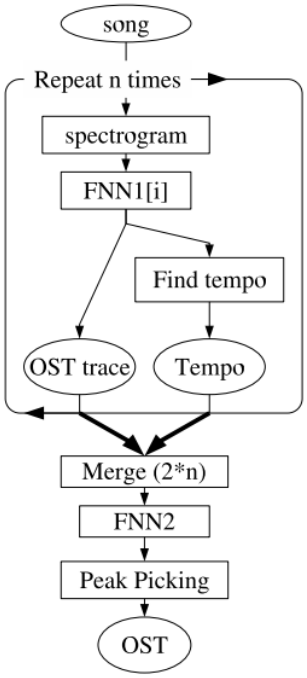


Figure 3 MANY-NETS Neural Network flow chat. (Lacoste and Eck 2005)

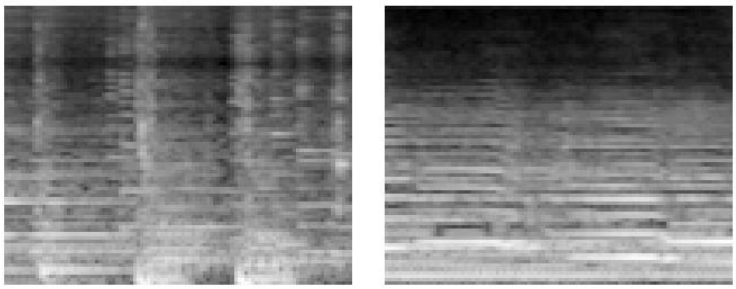
In the study by Eyben et al. (2010), the authors proposed a universal onset detection method employing Bidirectional Long Short-Term Memory (BLSTM) recurrent neural networks (RNNs). BLSTM enables the RNN to capture context by retaining the historical inputs for multiple epochs (i.e. training iterations). The preprocessing steps involve segmenting the audio signal into overlapping frames of sample lengths W = 1024 or W = 2048. Subsequently, the Short-Time Fourier Transform (STFT) is applied to generate a spectrogram, which is then transformed into the Mel spectrogram using a filter bank. Based on psychoacoustic knowledge, a logarithmic representation of the Mel spectrogram is employed to align with human perception of loudness.

In the neural network stage, an RNN with BLSTM architecture is utilised. Two log Mel spectrograms serve as inputs, and the network consists of three hidden layers in both directions, each with 20 Long Short-Term Memory (LSTM) units. The output layer, with two units representing the probabilities for "onset" and "no onset", adopts an activation function ensuring values between 0 and 1. During training, each audio sequence is presented in order and frame by frame, and performance is evaluated after each epoch.

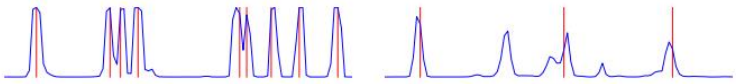
If no improvement occurs over 20 epochs, training is halted, and the network with the best performance is chosen.

In the work of Böck et al (2013), who co-wrote the Eyben et al (2010) study, an alternative approach to onset detection is presented. Rather than relying on handcrafted peak-picking algorithms, an RNN-trained algorithm is proposed for peak-picking, trained in a supervised manner. The optimisation caters to both online (real-time) and offline applications.

Inspired by the usage of Convolutional Neural Networks (CNN) to find edges in images, decided to train a CNN algorithm for onset detection on excerpts of spectrograms. CNNs are state-of-the-art Neural Networks for Computer Vision tasks but were previously used for other MIR tasks such as genre classification, tagging, key detection and chord detection with “promising results” as stated by Schlüter and Böck (2013). The system would find peaks in the spectrogram image itself, as opposed to spectrograms represented as sequential data for RNNs, which was state-of-the-art at the time. The CNN model proved to be more accurate than the state-of-the-art RNN, with an F-score of 0.885 against an F-score of 0.873 by the RNN on the Böck dataset. Schlüter and Böck (2014) then improved the CNN to reach an F-score of 0.903, becoming the current state-of-the-art for Onset Detection models.



(a) input spectrograms (mid-sized window length only)



(b) network output (blue line) and ground truth (vertical red bars)

Figure 4. Visualisation of the input and output of a CNN Onset Detector. (Schlüter and Böck 2014)

Chapter 3: **The Training Dataset**

As mentioned previously, there is no public dataset available for training Onset Detection models that focus on accents. Since recording hundreds – or possibly thousands – of musical snippets with different instruments would be very costly and time-consuming, a programmatic and digital approach to creating a training dataset.

3.1 The Music21 Library

Music21 is a Python-based toolkit for computer-aided musicology. Created by professors of music at MIT, it is an open-source library that can be used for analysing large datasets of music, editing musical notation, and composing music in an programmatic manner. Music21 parses MusicXML files and abstracts the contents of the file in an object-oriented representation, which makes programmatically editing MusicXML an easier and more intuitive process. Additionally, Music21 comes with a very large corpus of music in MusicXML and ABC format. The first version data generation pipeline will make use of the Music21 corpus, a selection of MusicXML tracks from the corpus will serve as input files to the transformation function of the data pipeline.

3.2 Transforming the Music21 tracks

3.2.1 The Böck Dataset

Sebastian Böck is a MIR researcher who made several remarkable contributions to the Onset Detection field, most notably by creating the most efficient Onset Detection algorithm using CNNs (Schlüter and Böck 2014), among others.

All of his works on Onset Detection seem to use the Böck Dataset as the training dataset. Since the aim of the project is to extend a preexisting Onset Detection model, we believed that it would be appropriate to generate a dataset resembling datasets used to train these models. Since Mr Böck courteously granted permission to download his dataset, we will create a data pipeline that will generate a dataset with a similar structure to his. The dataset is simple, containing three folders:

- An “audio” folder which contains all the audio snippets, in FLAC format
- An “annotations” folder contains a text file for each audio snippet with the second at which the onsets take place
- A “splits” folder, which contains a text file with the names of all the files in the “audio” folder, sometimes followed by a range, representing from which to which second the neural network must start reading the audio file

The structure of this dataset was extended by adding another set of files for each audio snippet, taking a record of each instant where an accented onset appears, similarly to the way onsets are recorded.

Excerpts of polyphonic music were also be considered to complexify the dataset. In the Böck dataset, polyphonic onsets, such as chords, are considered as a singular onset instead of multiple onsets played at the same time. Therefore, the same approach was taken when applying polyphonic snippets to the accented onset detection function. As all notes played at once are treated as one onset, either all notes would be accented, or none. Even though it is not a realistic representation of accents in polyphonic music,

this adaptation had to be made as accenting notes individually would break the logic of existing onset detection models, meaning that the complexity of the project would have to increase as a new model would have to be implemented to suit the structure of the dataset.

3.2.2 Preprocessing

Every piece was first pre-processed to ensure that each snippet was fit to go through the transformation function. This included setting the instruments to the piano, removing all repeats, removing any expressions and articulations, as well as creating the snippets themselves.

As the snippets of the Böck dataset have a duration of between 5 to 20 seconds, the length of the snippets from the corpus tracks have been made of similar length. However, each snippet has been assigned a different tempo, for the purpose of diversifying the dataset. The tempo is selected at random from the normal distribution, with a peak of 125 BPM and a standard deviation of 20, and inside the range of 60 to 240 BPM. These settings for randomness have been chosen since music is more often written in tempos of 125 BPM. Note that the BPMs are all be expressed in 4/4 time, because Music21 conveniently represents the time at which the note appears based on how many crotchets (or quarter notes) away the note is from the beginning of the piece, instead of based on the time signature.

The division of snippets has been made at the level of measures, and each resulting snippet has been used for the dataset.

3.2.3 The Transformation Function

Each object of type `music21.note` holds an attribute indicating how many quarter notes away it is situated from the beginning of the piece. This value is stored in the `offset` attribute. The onset of a note, expressed in seconds, is therefore calculated using this equation:

$$\frac{\boldsymbol{note.offset} \times \mathbf{60}}{\boldsymbol{tempo}}$$

Adding an accent is decided at random with a 20% probability, unless an accent is already present, in which case that accent is kept, and all other notes played at the same time are accented. Notes that are tied to a previous one are not recorded as onsets and therefore are not accented.

As note objects of different parts are held as elements of separate `music21.stream.base.Part` objects, iterating through every note in a polyphonic piece by order of appearance proved to be more complex, especially since it was important to make sure that all notes played at the same time were accented. A pointer system was implemented to ensure that the decision of accenting notes was made at one time for all notes playing at a given instant. Each pointer would be assigned to a part and set at the first note of the part. At each decision, only the pointers pointing to the note with the lowest `offset` value would be considered. Once the decision is made, the onsets are recorded in a dataframe, those pointers are moved to their next note. If there is no next note to point at, the pointer is destroyed. The process repeats itself until all pointers are destroyed, in which case the transformation function is complete.

Once the transformation is complete, the dataframe holding record of the onsets is processed and writes two text files for the snippet: one that records all instances of onsets, one that records all instances of accents.

A multithreaded approach was taken when writing the transformation function. Each transformation is being computed by a different python process, which significantly speeds up the task, as opposed to performing each transformation sequentially in a single process.

Code Snippet 1: The `Pointer` class and its associated functions

```
import music21 as m21

( ... )

class Pointer:
    def __init__(self, notes, offset):
        self.notes = notes
        self.index = 0
        self.offset = offset

    def update_index(self, index):
        self.index = index
        if len(self.notes) > index:
            self.offset = self.notes[index].offset
            return True
        else:
            return False

    def get_note(self): return self.notes[self.index]

    def get_index(self): return self.index

    def get_offset(self): return self.offset


def make_pointer_dict(piece):
    pointer_dict = {}
    for i in range(len(piece.parts)):
        if len(piece.parts[i].flatten().notes) > 0:
```

```
            pointer_dict[i] =
Pointer(piece.parts[i].flatten().notes,
piece.parts[i].flatten().notes[0].offset)
    return pointer_dict



def get_lowest_pointers(pointer_dict):
    # returns indexes of pointers with the lowest offset
    low = min([pointer_dict[i].get_offset() for i in
pointer_dict])
    return [i for i in pointer_dict if
pointer_dict[i].get_offset() == low]



def move_pointers(pointer_dict):
    to_move = get_lowest_pointers(pointer_dict)
    for i in to_move:
        if
pointer_dict[i].update_index(pointer_dict[i].get_index() + 1)
== False:
            del pointer_dict[i]
    return pointer_dict
```

Code Snippet 2: The `transform()` function and the process of accenting notes.

```
import music21 as m21

( ... )

def transform(snippet):

    ( ... )

    pointer_dict = make_pointer_dict(snippet)

    # inner function for accenting notes
    def accent(ptrs):
        for i in ptrs:
            note = pointer_dict[i].get_note()
            if note.tie == None or note.tie.type == 'start': # 
do not accent if note is tied
                note.articulations = 
[m21.articulations.Accent()]

    # iterating through every note in snippet
    while len(pointer_dict) > 0:
        lowest_ptrs = get_lowest_pointers(pointer_dict)

        # if any note in lowest_ptrs is already accented, 
accent the rest of the notes in lowest_ptrs
        if any(m21.articulations.Accent in note.articulations 
for note in (pointer_dict[i].get_note() for i in 
lowest_ptrs)):
            accent(lowest_ptrs)
        # accent notes
        elif random.random() < (0.2):
            accent(lowest_ptrs)

        ( ... )

        pointer_dict = move_pointers(pointer_dict)

    ( ... )
```

3.3 Generating Audio Files

An important issue that arose while generating audio files directly using the functions provided by Music21 was that the accents were not present in the output audio. This was an instant setback for the data pipeline since the whole purpose of it was to create music snippets with instances of audible accents.

To tackle this issue, we have instead opted to export the generated tracks in MusicXML format and use the music notation software MuseScore4 to convert the MusicXML files to FLAC files. Using MuseScore4's command line options, the data pipeline generates and run batch files once all of the MusicXML files were converted. Similarly to the transformation function, the conversion task was divided into multiple processes for a faster execution time.

3.4 The First Generated Dataset

For the first version of the dataset 20 MusicXML files were selected from the corpus included in the Music21 library. All 20 files were preprocessed to have only the piano instrument playing, and the maximum number of possible snippets were taken from each file. In total, this resulted in a dataset composed of 353 audio files.

When training the onset detection model mentioned in section 4.1 using the generated dataset, ridiculously high results would come out during evaluation. Results would average a precision score of 1.00, recall score of 0.90, and an F-score of 0.95. However, when evaluating on the Böck dataset, the model would record a lot of false negatives, resulting in a very good precision score, in the high 0.90's, but a concerningly low recall score, around the 0.20's. This means that while the model very rarely falsely marks non-onset instances as onsets, it overlooks a lot of onsets and falsely considers them as non-onsets.

This could be the result of many different factors concerning the homogeneity of the snippets in the generated dataset. The most obvious difference is that the tracks using the same digital instrument, when the Böck dataset mostly consists of snippets played by various physical instruments. The Böck dataset is also made from a range of different genres and styles of music, whereas the Music21 corpus mostly contains older choral and classical pieces, as they aren't protected by copyright law. This could mean that the model only considered onsets represented by a signal of a piano sound as an onset. Another reason could be that all of the snippets are being played in the same dynamic, which could result in the model being biased against notes that are played in a different dynamic, and not consider them as onsets.

This version of the dataset was therefore discarded and a more improved version of the dataset, detailed in the section below.

3.5 The Second Version of the Dataset

3.5.1 Downloading MusicXML files from MuseScore.com

To create a more diversified dataset, I have made use of the wide range of musical compositions available on the online score-sharing platform MuseScore.com. The website offers the option to directly download scores in MusicXML format, making it easy for me to integrate into the data pipeline.

Close to 90 MusicXML files were downloaded, all with at most a creative commons license, and made sure to choose compositions using varying instruments, of varying genres, tempos, dynamics and time signatures, to diversify the dataset as much as possible. The scores utilised in this project are listed in Appendix A.

3.5.2 Additional Preparation and Preprocessing on the MusicXML files

As the downloaded MusicXML files from MuseScore.com were much more complex than the files from the Music21 corpus, additional preprocessing steps were needed to be taken in order to make the MusicXML file more appropriate to the data pipeline and discard any excessive complexity in the dataset.

Firstly, any elements that would cause irregularity between when the onset is recorded to occur and when the onset actually occurs are removed. This includes all repeats, such as repeat bars, codas and signs, fermatas, accelerandos and rallentandos and any other tempo changes. Each measure was also marked with their appropriate time signature, as it is possible for a measure to have more or less notes than it is assigned to have. Secondly, all elements that create ambiguity about whether an onset occurs or not were also removed. This includes, glissandos, trills, grace notes and arpeggios.

Code Snippet 3: Extract of the `clean()` function in data-pipeline/preprocessing.py – setting the appropriate time signatures for each measure.

```
import music21 as m21

( ... )

def clean(piece : m21.stream.Score):

    for part in piece.getElementsByClass(m21.stream.Part):

        actualTimeSignature = None
        for measure in
part.getElementsByClass(m21.stream.Measure):
            barDuration = measure.highestTime
            if measure.timeSignature != None:
                actualTimeSignature = measure.timeSignature
            if barDuration !=
actualTimeSignature.barDuration.quarterLength:
                n = 2
                t = barDuration
                while floor(t) != t:
                    t = t * 2
                    n += 1
                measure.timeSignature =
m21.meter.TimeSignature(f"{int(t)}/{2**n}")
            else:
                measure.timeSignature =
m21.meter.TimeSignature(f"{actualTimeSignature.numerator}/{act
ualTimeSignature.denominator}")

    ( ... )

    return piece
```

Unfortunately, a lot of the downloaded files also needed to be manually prepared in order to make them compatible with the data pipeline. MusicXML allows for different voicings in a part. 2 to 4 layers of notes could be layered at the same position in a measure. When Music21 exports such files back into MusicXML, it confuses the voicings and appends all notes into one voicing in the outputted file. This results in a corrupted MusicXML file with very irregular measures, which causes problems especially in scores with multiple parts, as the same measure would be of different length for each part. The MusicXML file therefore needed to be manually prepared using the MuseScore4 software, by separating each voicing and writing them into a separate part.

As a significantly larger number of files are used as inputs to the data pipeline, only up to 4 snippets are selected for the dataset. Setting the tempo from the snippet

was changed to a value ranging between below and above 35 BPM from the original score, to prevent from snippets being inhumanely fast or ridiculously slow.

Code Snippet 4: Extract of the `clean()` function in data-pipeline/preprocessing.py – example of how an object would be removed during preprocessing

```
import music21 as m21

    ( ... )

def clean(piece : m21.stream.Score):

    ( ... )

    for part in piece.getElementsByClass(m21.stream.Part):

    ( ... )

        for measure in
part.getElementsByClass(m21.stream.Measure):

           ( ... )

            # remove repeat
            for repeat in
measure.getElementsByClass(m21.bar.Repeat):
                measure.remove(repeat)

            for note in
measure.getElementsByClass(m21.note.Note):

                # remove all expressions, includes Fermata
                note.expressions = []
                note.articulations = []

                # remove grace notes
                if note.duration.isGrace:
                    measure.remove(note)

        ( ... )

        # remove glissando
```

```
        for gliss in
part.getElementsByClass(m21.spanner.Glissando):
            part.remove(gliss)

        ( ... )

    return piece
```

3.5.3 The Resulting Dataset

The generated dataset consisted of 313 WAV files. I have chosen to export the snippets in WAV format instead of FLAC, as some FLAC files presented a lot of clipping whereas the WAV files did not.

The dataset features over 30 different instruments, the most frequently appearing instruments being the piano, electric and acoustic guitar, electric bass, violin and trumpet. The choice of these instruments as the most frequently appearing instrument is due to their popularity in the world of music.

There is a decent variety of number of parts a snippet has in the dataset, with the most frequent number of parts ranging from 1 to 5. Please note that a part does not necessarily represent a monophonic signal, as a chord could exist inside the part. Additionally, some parts could contain significantly less notes than others.

The dynamics of each snippet is also quite fairly distributed, although a lot of the MuseScore compositions chose to not include a dynamic, therefore the “default” dynamic is predominant.

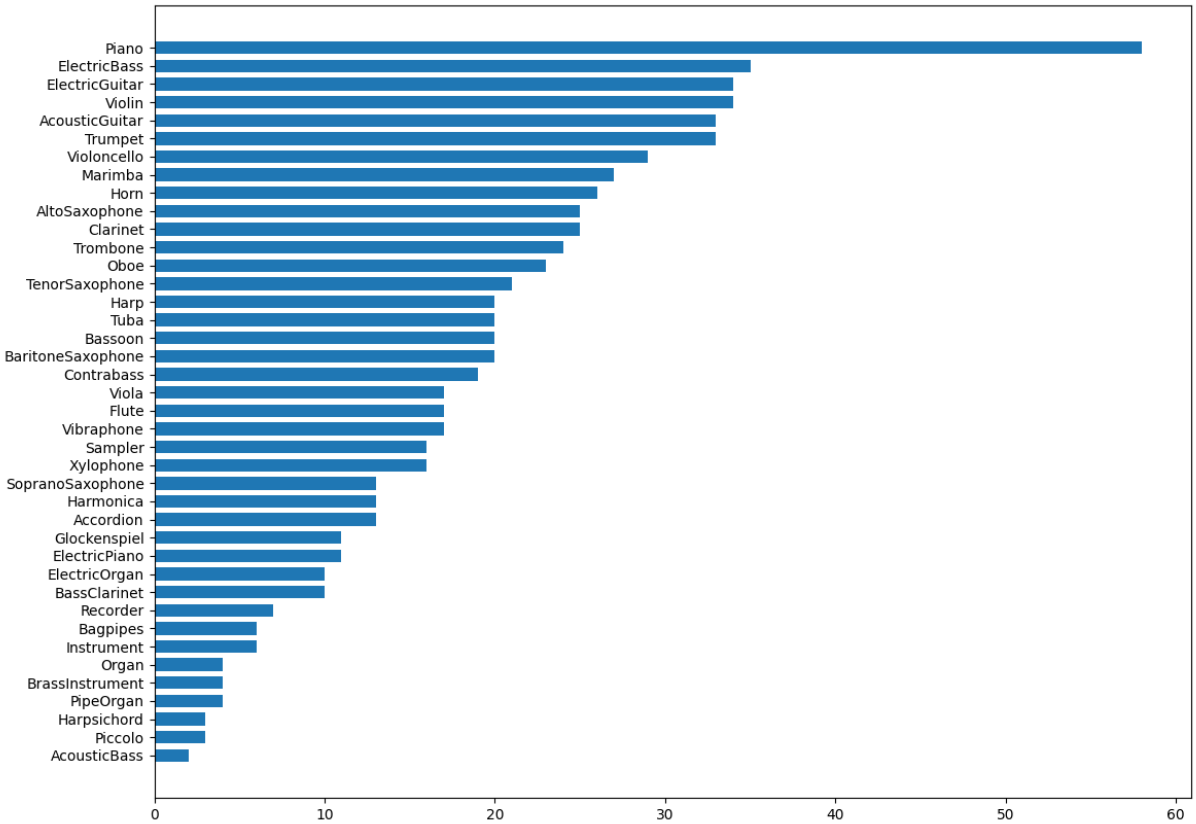


Figure 5. Distribution of audible instruments appearing in the dataset.

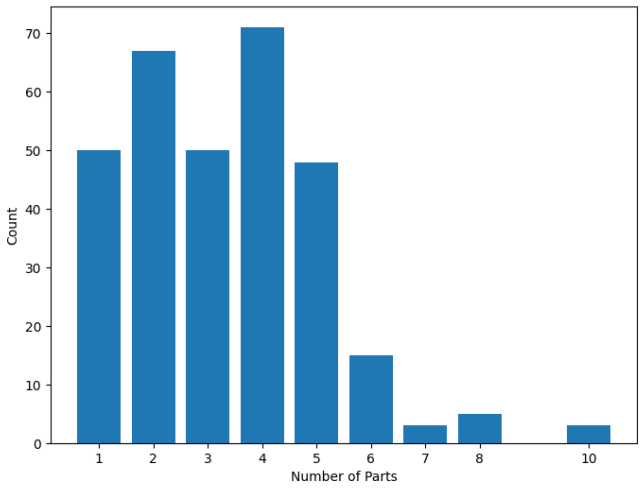


Figure 6 Distribution of the number of parts in a snippet in the dataset.

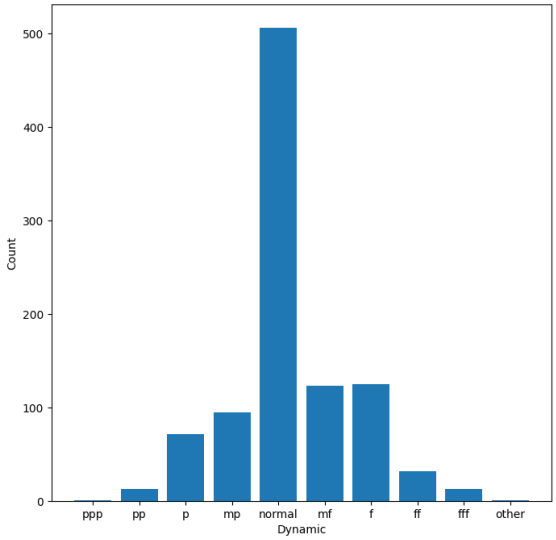


Figure 7. Distribution of dynamics observed across the dataset.

Additionally, MuseScore4 adds between 2 to 3 seconds of additional time to the generated audio files to let the last note “ring out”, hence why a few tracks appear to be longer than 20 seconds, and all tracks have a couple of seconds of silence before they end. This could mislead the model into assuming that onsets never happen in the last seconds of a snippet and be biased against any event happening in the last seconds.

3.5.4 Potential Improvements to the Dataset

As observed, a lot of different categories tend to dominate in the dataset. As a result, the trained model could potentially be biased for or against snippets with or without certain characteristics. A few modifications to the data pipeline could be made in the future in order to improve the quality of the generated dataset. These include:

- Randomising the instrument that plays the notes in the snippet, to ensure that the distribution of instruments is near even.
- Randomising the dynamics for each snippet, ensuring that the distribution of dynamics is near even.
- Cutting out the last 2 or 3 seconds of each resulting WAV file or using a tool that better converts MusicXML into FLAC files, as they take less space than WAV files.
- Adding more input MusicXML files to the data pipeline, generating a larger and more robust training dataset.

Chapter 4: **The Accent Detection Model**

4.1 The Chosen Onset Detection Model

The state-of-the-art Onset Detection models presented by Schlüter and Böck (2014) are not open source. Instead, another open source, documented, and most importantly efficient onset detection model had to be chosen for the project. Lindqvist (2019) has worked on reproducing the RNN model described by Böck, Schlüter and Widmer (2013) and CNN model described in Schlüter and Böck's (2013, 2014) papers. Both models are open-source and documented in Lindqvist's bachelor's thesis. Lindqvist states that he was not able to achieve the same F-scores as the original models – the trained CNNs obtained an average F-score of 85.5%, about 5% less than about 5% less than Schlüter and Böck's 90.3%, whereas the RNNs obtained an average of 86.3%, only 1% less than Schlüter and Böck's. Although the author states that his models aren't exact replicas of the original models due to a few missing details in the original papers that had to be compensated for with "educated guesses", the scores of the model were deemed satisfying enough to be used for the Accent Detection model. I decided to focus on working on the CNN, not only because it produced better results, but it was also significantly faster to train: 1 epoch of the CNN takes around 1:30 minutes to complete, whereas 1 epoch of the RNN takes around 24 minutes to complete.

The following subsections will briefly describe the structure of Lindqvist's (2019) CNN-based onset detection model.

4.1.1 Pre-processing

First, the signal is resampled into mono 44.1 kHz. From the resampled signal, three magnitude spectrograms are calculated, each with the same hop size of 10 ms, and with different windows sizes of 11.61, 23.22 and 46.44 ms. The spectrograms are then filtered using 80-band Mel filters from 27.5 Hz to 16 kHz.

The shape of the data becomes a 3-dimensional vector of size fx80x3, where f is the number of 10 ms frames in the audio. A 15-frame wide window would slide over the audio data, which meant 7 frames of padding needed to be added at the beginning and at the end of each file.

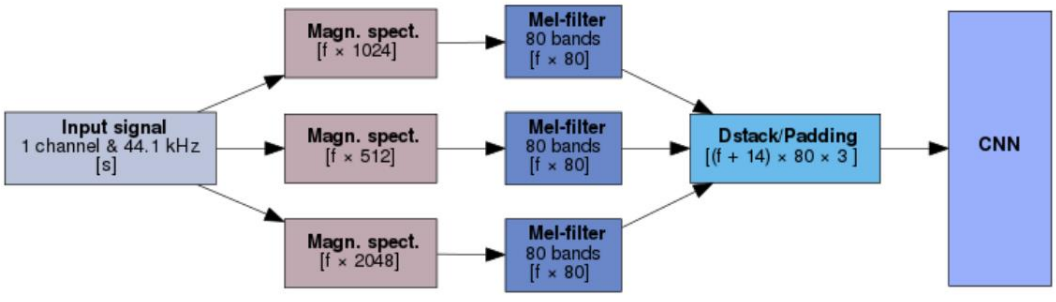


Figure 5. Preprocessing steps for the CNN (Lindqvist 2019)

The y vector of the model would be of length f. Each frame is allocated a scalar, either 0 or 1, representing the existence of an onset in that frame: 1 would mean that there is an onset, 0 that there isn't.

4.1.2 The Onset Detection Function

The model itself was written in Python, using the Keras library. It closely follows the architecture described by Schlüter and Böck (2014). The model consists of 8 layers, ordered as below:

1. A 2D Convolutional Layer, with 10 filters and a kernel dimension of 7x3. It takes 3 input channels of size 15x80 as input. Its activation function is ReLU.
2. A 2D Max-Pooling Layer, for down-sampling the input. This is done by taking the maximum values of input windows of size 1x3 for each channel of the input.
3. A second 2D Convolutional Layer, with 20 filters and a kernel dimension of 3x3. It takes 10 input channels of size 26x10. Its activation function is also ReLU.
4. A second Max-Pooling Layer of the same size as the first one
5. A Dropout layer of rate 0.5. This layer randomly sets input units to 0 with a frequency of 0.5 at each step during training time, which helps prevent overfitting.
6. A Flatten layer, which flattens the multi-dimensional input into a one-dimensional array.
7. A first Fully Connected layer, which condenses the input into 256 sigmoid units.
8. A second Fully Connected layer, which condenses the input into the single sigmoid output unit.

The model makes use of the SGD optimiser, with a learning rate of 0.05, but with a fixed momentum of 0.8, unlike the original implementation, which makes use of a linearly increasing momentum from 0.45 to 0.9 between epochs 10 and 20. This decision was made due to Lindqvist's limited resources and need for stopping and resuming the training process. He states that hyper parameter scheduling works poorly in combination with resumption in Keras and argues that keeping a fixed momentum value of 0.8 is a good compromise. Since the computing resources used for this project are also limited, the fixed momentum was kept as the ability to stop and resume training proved to be time efficient.

The model is trained with the 8-fold cross validation technique: the dataset is divided in 8 parts, or folds, and 8 versions of the model are trained, with each model using a different fold for the validation and evaluation sets, and the remaining 6 folds as the training dataset. The results of each model are averaged during the evaluation process. That way, the entirety of the dataset is covered during training while keeping the training, validation and evaluation sets different. Again, due to limited computing resources, the models were trained for 20 epochs, instead of training for 100 epochs like Schlüter and Böck did.

A trained model would output a prediction for any given audio snippet. The prediction is a vector of length f, giving a value between 0 to 1 for each frame. That value represents the probability of an onset existing in the frame – a value closer to 1 represents a higher likelihood of observing an onset.

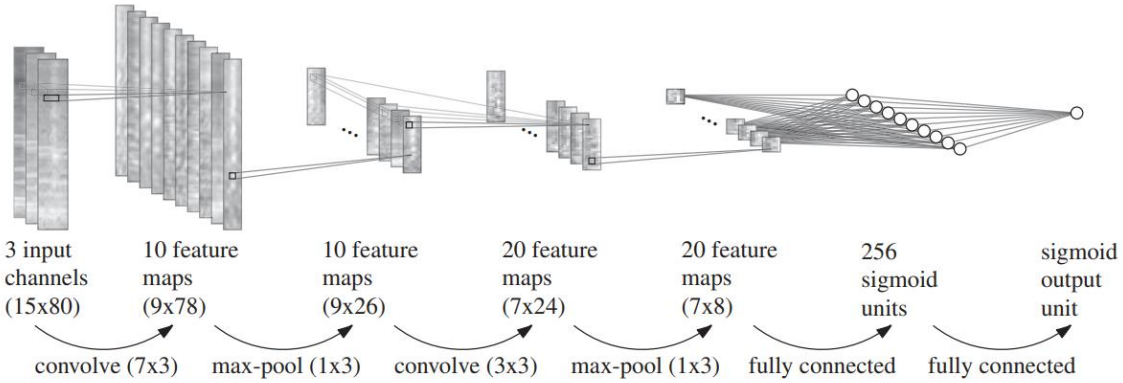


Figure 6 Graphical Representation of the Onset Detection Function by (Schlüter and Böck 2014)

4.1.3 Peak-picking

The peak-picking function provided by the Madmom library was used in the CNN for postprocessing. The peak-picking function accepts values over a certain threshold as onsets and performs peak-picking on the detected onsets. The function outputs a vector consisting of the indexes of the frames that contain an onset. Lindqvist has set the threshold to 0.54 but mentioned that lowering the threshold could balance out the precision and recall results of the model.

4.1.4 Evaluation

The model is evaluated using three metrics: precision, recall and f-score. Precision is the ratio of values correctly predicted as positive with all of the values predicted as positives. Recall is the ratio of values correctly predicted as positive with all of the values that are truly positive. F-score is calculated as the harmonic mean of precision and recall.

When evaluating the model, only results concerning onsets are taken into account – frames correctly predicted as onsets (true positives), frames incorrectly predicted as onsets (false positives) and frames incorrectly predicted as non-onsets (false negatives). Metrics concerning correctly predicted non-onsets are considered irrelevant, as the ratio of non-onsets compared to onsets in each snippet is so high, that including these results would skew the results, and fail to give an appropriate picture of how well the model performs when detecting onsets.

The evaluation function allows a margin of error of up to 3 frames. Onsets that are guessed to occur within 3 frames after the true value are considered as positive guesses.

4.2 Modifying the Model for Accent Detection

4.2.1 Modelling the Target Vector y

As the modification will be done for identifying two types of notes, a single-dimensional y vector becomes unsuitable. Each frame would have to represented by multiple probabilities to represent the three possibilities: non-accented onsets, accented onsets, and non-onsets. Three options for formatting the vector were explored:

1. As a one-hot tensors, for multiclass classification. We define three classes, non-onset, non-accented onset, and accented onset, and each frame is assigned one single class. The target tensor would take the shape of fx3, with a value of 1 the position of each corresponding class, and 0 in the other two. There would be then three possible values for a given frame:

- `[ 1, 0, 0 ]` for a non-onset
- `[ 0, 1, 0 ]` for a non-accented onset
- `[ 0, 0, 1 ]` for an accented onset

Code Snippet 5: Code for formatting y as a one-hot tensor for multiclass classification

```
import numpy as np
from tensorflow import one_hot

( ... )

def preprocess_y(anns, n_frames):
    # 14 frames of padding are added in preprocess_x
    n_frames -= 14
    for i in range(2):
        for j in range(0, len(anns[i])):
            anns[i][j] = (anns[i][j] * 100).astype(int) # convert to ms

    accents, onsets = anns[1].astype(int), anns[0].astype(int)

    categories = []
    for i in range(n_frames):
        if i in accents: categories.append(2)
        elif i in onsets: categories.append(1)
        else: categories.append(0)
    y = one_hot(indices=categories, depth=3, on_value=1, off_value=0)

    return y
```

2. Tensors for multilabel classification. The difference between the y tensors for multilabel and multiclass classification are that each input can belong to multiple classes or none. Following this structure, we will define two labels: onset and accent. All frames with an onset, accented or not, are assigned the onset label, frames with accented onsets are assigned with the accent label, and frames with no onsets are assigned no labels. The target tensor would take the shape of fx2, and a frame could take either one of the three possible values:
    - `[ 0, 0 ]` for a non-onset
    - `[ 1, 0 ]` for a non-accented onset
    - `[ 1, 1 ]` for an accented onset

Code Snippet 6: Code for formatting y for the 1st version of multilabel classification

```
import numpy as np
```

```
( ... )

def preprocess_y(anns, n_frames):
    # 14 frames of padding are added in preprocess_x
    n_frames -= 14
    for i in range(2):
        for j in range(0, len(anns[i])):
            anns[i][j] = (anns[i][j] * 100).astype(int)

    accents, onsets = anns[1].astype(int), anns[0].astype(int)

    y = np.zeros((n_frames, 2))
    for i in onsets: y[i][0] = 1
    for i in accents: y[i][1] = 1

    return y
```

3. As the third option, an alternative labelling system for a multilabel tensor was chosen. The two labels are non-accented onset, and accented onset. A frame could then take one of the below values:
    - `[ 0, 0 ]` for a non-onset
    - `[ 1, 0 ]` for a non-accented onset
    - `[ 0, 1 ]` for an accented onset

Code Snippet 7: Code for formatting y for the 2nd version of multilabel classification 2nd

```
import numpy as np

( ... )

def preprocess_y(anns, n_frames):
    # 14 frames of padding are added in preprocess_x
    n_frames -= 14
    for i in range(2):
        for j in range(0, len(anns[i])):
            anns[i][j] = (anns[i][j] * 100).astype(int)

    accents, onsets = anns[1].astype(int), anns[0].astype(int)

    y = np.zeros((n_frames, 2))
    for i in onsets:
        y[i][0] = 1
```

```
    for i in accents:
        y[i][1] = 1
        y[i][0] = 0

    return y
```

Three different models using each one of the three different target vectors were created and were trained over the same folds of the dataset to determine which target vector format allowed for the better results. The model using the first option will be referred to as the Multiclass Model, the model with the second option will be referred to as the Multilablel-1 Model, and the model with the third option will be referred to as the Multilabel-2 Model.

4.3 Training

The procedure taken for training was no different from the one taken by Lindqvist for his model. All three of the models were trained using 8-fold cross-validation with 20 epochs. As the dataset consisted of 312 files, each fold contained exactly 39 files, close to the size of the folds for the Böck dataset, which is between 40 and 41. After each epoch, the model was saved as an HDF5 file. The models at the epochs at which the loss on the training set and the loss on the validation set were the lowest were also saved.

Chapter 5: **Evaluation**

5.1 Post-processing the Predictions

The peak-picking function in the Madmom library is designed for output vectors with values representing the probability of observing an onset or not. The Multiclass model was therefore considered unsuitable for the peak-picking function, as the values in the prediction represent a fundamentally different prediction. The first value represents the probability of observing a non-onset in contrast to a non-accented onset or accented onset. The second value represents the probability of observing a non-accented onset in contrast to a non-onset and accented onset. The absence of an implicit divide between onset and non-onset renders model to be incompatible with the peak-picking function. The Multiclass Model was therefore discarded from the evaluation.

Furthermore, a snippet expressed as a one-hot tensor highlights the class imbalance, which resulted in the predictions being heavily biased towards the non-onset class. Even if a prediction for any other class is high, the prediction for the non-onset class would generally still be higher.

For the Multilabel1 model, the peak-picking function was only applied on predictions of the ‘onset’ label, as it covers the entirety of the onsets, and the ‘accent’ label represent the presence of an accent itself as opposed the presence of an accented model. Predictions for the ‘accent’ label were instead combined based on the predictions for the ‘onset’ label. If for any neighbouring predictions at least one of the frames has been predicted to contain an onset, the neighbouring predictions were combined to those specific frames.

Lindqvist suggested lowering the peak-picking threshold from 0.54 to potentially balance out the evaluation results, so the threshold was lowered to 0.5, which indeed proved to produce slightly better results. A lower threshold of 0.15 was chosen for ‘accent’ labels as predictions for that label were considerably lower.

Code Snippet 8: Postprocessing function for Multilabel1

```
import numpy as np
from Madmom.features.onsets import peak_picking

( ... )

def postprocess_y(y_ons, y_acc, thresh_ons, thresh_acc):
    onsets = peak_picking(y_ons,
                          threshold = thresh_ons,
                          smooth = 5,
                          pre_avg = 0, post_avg = 0,
                          pre_max = 1.0, post_max = 1.0)

    accents = []
    for i, activ in enumerate(y_acc):
        if activ >= thresh_acc:
```

```
            accents.append(i)

    neighbours = []
    to_remove = []
    for i in range(1, len(accents)):
        if accents[i-1] == accents[i] - 1:
            if len(neighbours) == 0:
                neighbours.append(accents[i-1])
            neighbours.append(accents[i])
        elif len(neighbours) > 0:
            ons = [v for v in neighbours if v in onsets]
            if len(ons) > 0 :
                for v in neighbours:
                    if v not in ons:
                        to_remove.append(v)
            neighbours = []

    for v in to_remove:
        accents.remove(v)

    onsets = np.asarray(onsets)
    onsets = onsets.astype(np.float64) / 100.0

    accents = np.asarray(accents)
    accents = accents.astype(np.float64) / 100.0

    return onsets, accents
```

For the Multilabel2 model, the peak-picking function was applied on both 'non-accented onset' and 'accented onset' labels, with the same threshold of 0.5 for both labels.

5.2 Performance of Each Model

Each model was evaluated using the same three metrics as the original model: precision, recall and f-score. The results of all 8 models at their epoch with the lowest loss on their validation set were summed and the scores were expressed as one.

The model with the overall highest achieved metrics is Multilabel1. The average scores for all labels of all 8 Multilabel1 models are – precision: 0.753, recall: 0.635, f-score: 0.689. The scores for the onset label are – precision: 0.903, recall: 0.707, f-score: 0.793. The scores for the accent label are – precision: 0.248, recall: 0.282, f-score: 0.264.

Upon inspecting the outputs of the Multilabel2 model, it was found that most of the guesses for both classes were similar. This could be due to the fact that since the shape of onset and accented onset signals are too similar, and therefore can't be classed as a separate label. The Multilabel1 model's structure was therefore determined to be the most suitable for the Accent Detection task.

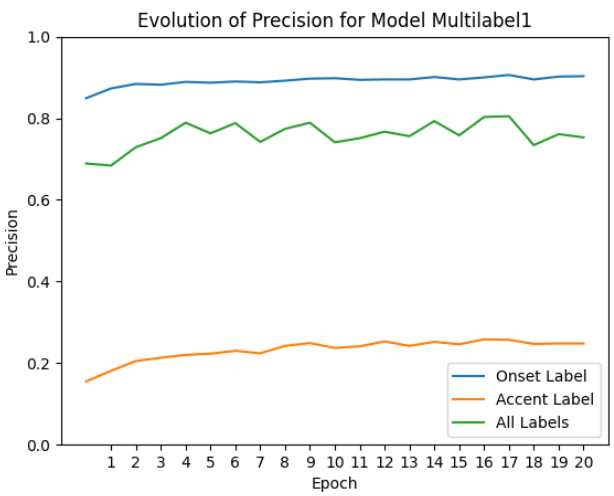


Figure 8. Evolution of the precision metric for the Multilabel1 model

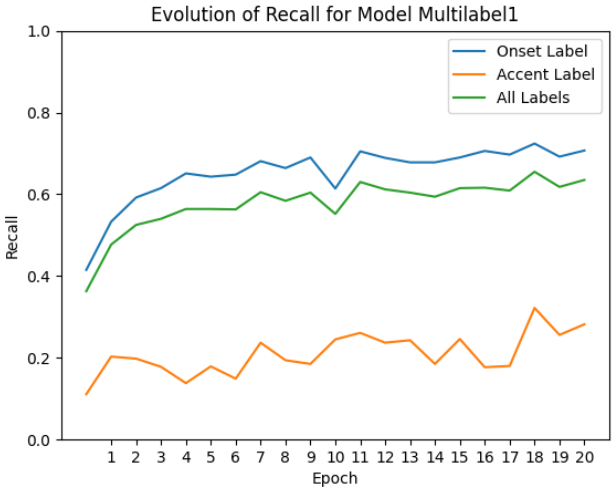


Figure 9. Evolution of the recall metric for the Multilabel1 model

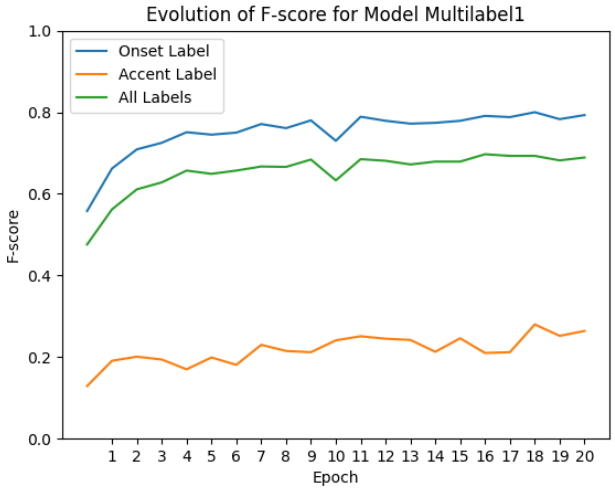


Figure 10. Evolution of the f-score metric for the Multilabel1 model

The progress of the models becomes very linear after the 3rd epoch. In addition, the models with the lowest loss on the validation set aren't necessarily the models trained at the last epochs. This could be due to the fixed momentum and learning rate of the SGD optimiser. Introducing scheduling to the momentum and/or learning rate hyperparameters could help prevent overfitting and improve the performance of the models. Scheduling adapts the value of the hyperparameters over time, which can enable the optimiser to make more effective hyperparameter updates for minimising the loss value.

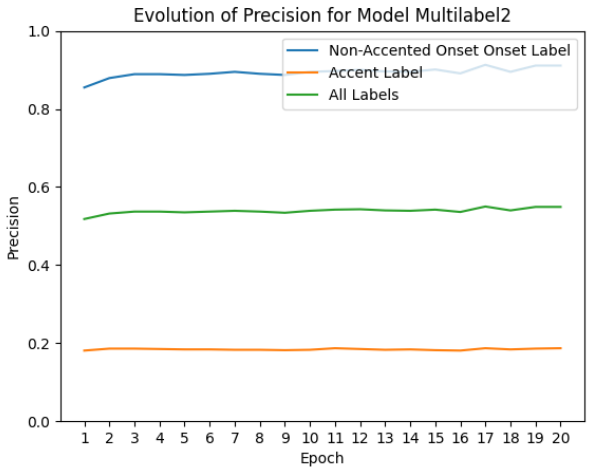


Figure 11. Evolution of the precision metric for the Multilabel2 model

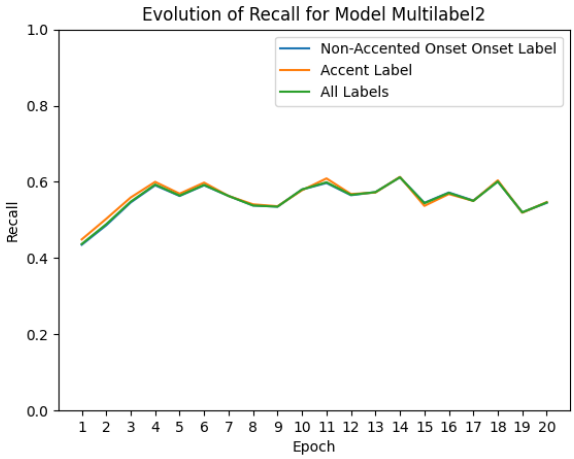


Figure 12. Evolution of the recall metric for the Multilabel2 model

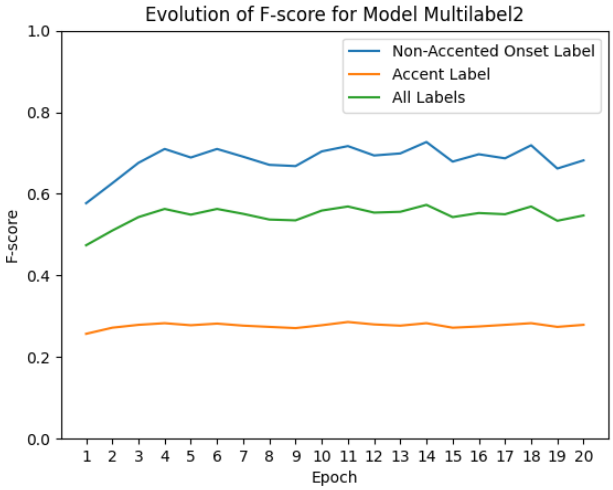


Figure 13. Evolution of the f-score metric for the Multilabel2 model

5.2.1 Performance on the Accent Label

While predictions for the 'onset' label were very good, the predictions for the 'accent' label were underachieving. In general, the prediction values for the 'accent label' were substantially low compared to the predictions for the 'onset' label.

An observation made about the model that could possibly explain the low results is the size of the convolution window. The original model has a convolution window size of 7, meaning predictions are only made based on the data appearing in 7 consecutive frames, or 70 milliseconds, which would only be able to cover at most 1 onset at a time. Since accented notes are fundamentally stronger than regular notes, it would be logical to make a decision on whether a note is accented based on the strength of the other notes in the input musical snippet. Therefore, a neural network starting with a convolution layer with a convolution window size large enough to cover few onsets at a time, could potentially produce more favourable results when it comes to detecting accented onsets.

5.3 A Deeper Dive into the Predictions

In order to better understand the model and its predictions, a few analyses were conducted based on snippets with different characteristics, with the hope of finding any

trends that may occur in the model's predictions. The analyses below have been conducted on results of the best-performing model, Multilabel1, and the predictions utilised for the analysis are those computed by the models with the lowest loss on their validation set have.

5.3.1 Performance of the Model based on the Number of Parts

The first analysis looked at how well the model performs with detecting onsets and accents based on the number of the parts in the snippet. The lowest precision score for onset detection was found in snippets with only 1 part. This came to us as a surprise as generally it is believed that onset detection on monophonic signals is easier, as there is a clearer distinction as to when an onset is heard. It is however important to note that only 15% of the dataset contains monophonic snippets, which could explain why the model could perform better with polyphonic snippets.

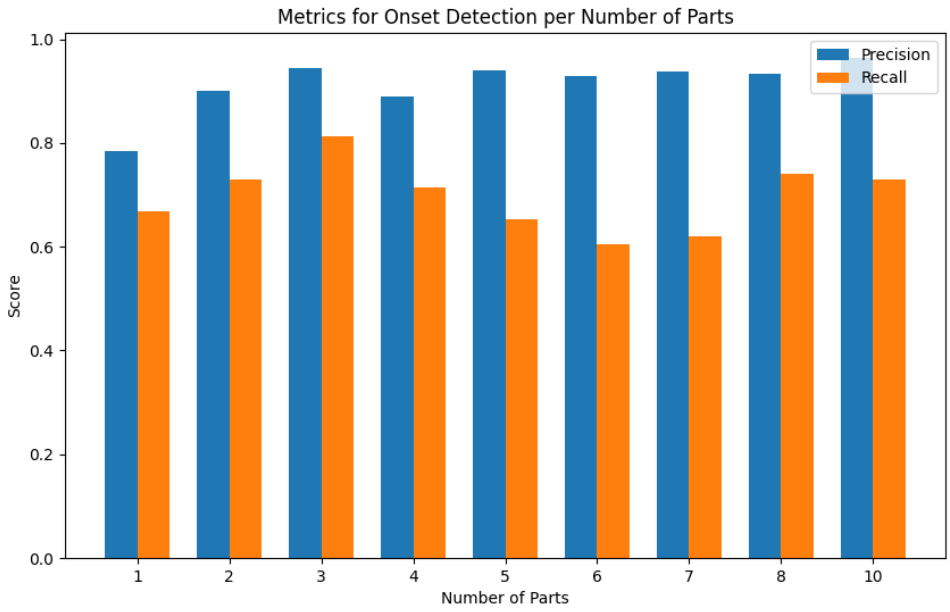


Figure 74. Comparison of Onset Detection metrics for snippets with different amount of parts

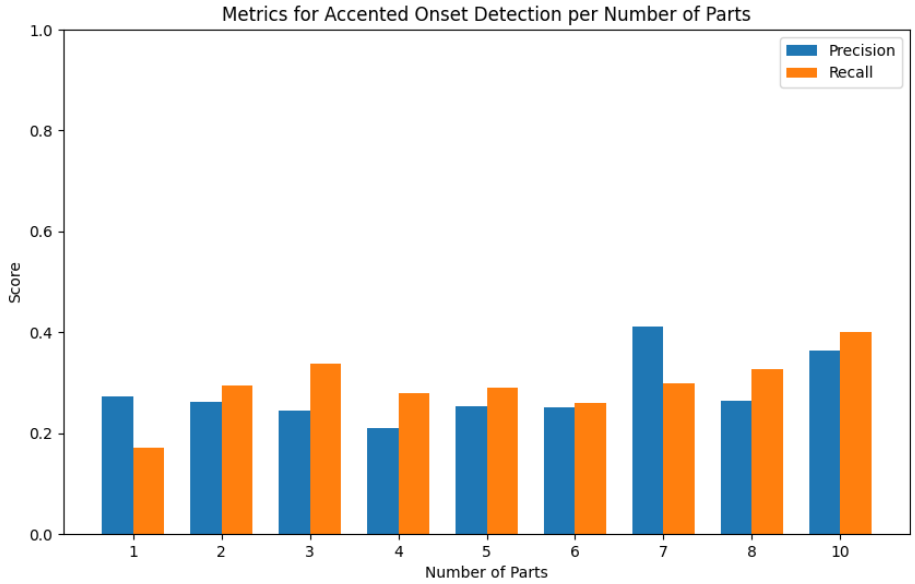


Figure 85. Comparison of Accent Detection metrics for snippets with different amount of parts

5.3.2 Performance of Accent Detection based on Dynamics

A factor that would most probably influence the predictions of accented onsets is the dynamic at which the note is intended to be played. As an accented note is a note that is louder than other notes, introducing a multitude of dynamics adds a significant amount of diversity into the dataset, as it guarantees that accented notes are played with a different loudness in different snippets.

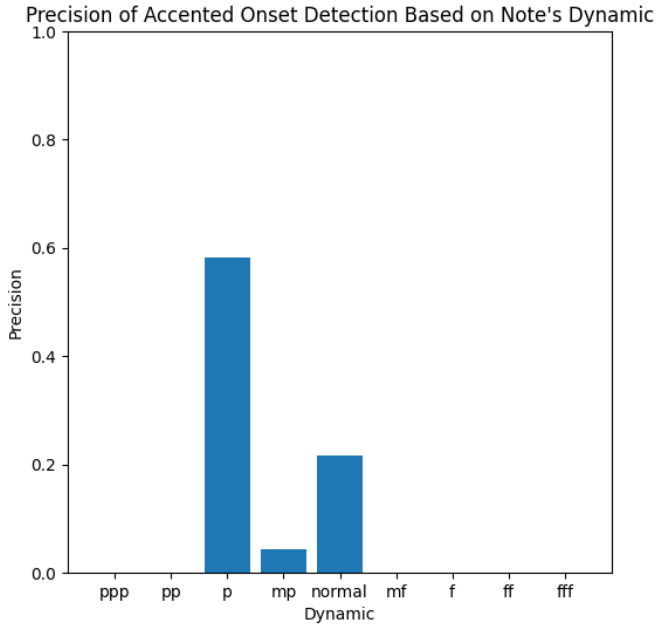


Figure 93. Precision of accented onset detection based on the note's dynamic

The evaluation was only conducted on monophonic signals, as it is possible for two onsets played at the same time to be played with different dynamics. We therefore must take the very small sample size into consideration when analysing the results.

The model was only capable of detecting accents played in dynamics ranging from piano to the normal dynamic. All accents played in the mezzo forte dynamic or louder were falsely predicted as non-accents.

5.3.3 Performance of Accent Detection Based on Pitch

The last analysis was conducted on the precision of accent detection based on the accented note's pitch. The notes were grouped in octaves, which is how pitch classes are separated in Western music. For a similar reason to the analysis based on the dynamic of the note, only monophonic snippets were considered. The definition of polyphonic music is two different melodies playing at the same time, thus enabling the inevitability of two onsets with different pitch frequencies played at once.

The model performed best at detecting accents in the 3rd, 4th and 5th octaves, which are the most commonly used octaves in music. For this reason, the dataset consisted mostly of notes played in the aforementioned octaves, creating a significant disparity in notes played at different frequencies. Amongst monophonic snippets, only 36 out of 267 accented notes were played in the 1st, 2nd or 6th octaves.

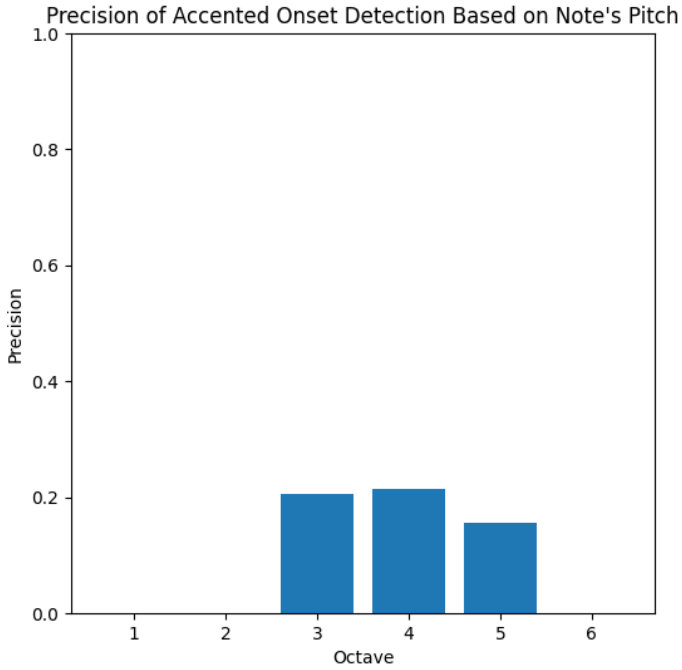


Figure 106. Precision of accented onset detection based on note's pitch.

Chapter 6: **Conclusion**

6.1 Project Conclusion

The project aimed at creating a CNN model for detecting instances of onsets and accented onsets in a musical snippet, based on the architecture of a state-of-the-art Onset Detection model. Since no known research has been made on this subject, no datasets relevant to Accent Detection exist. The project therefore also consisted of programmatically generating a dataset consisting of musical snippets with randomised instances of accents.

The generated dataset contained 312 snippets of length between 5 and 20 seconds, with both monophonic and polyphonic music, and featuring over 30 different digital instruments.

The Onsted Detection model has been mainly modified by changing the shape of its target tensor, different shapes and classification types have been tested, and multilabel classification has been found to be the most appropriate and effective for the task.

The task of automatically detecting accented onsets based on a musical snippet input proved to be difficult to achieve: while results for classifying onsets in general were still satisfying, the model greatly struggled with detecting accented onsets, with a precision of 0.248, recall of 0.282 and an f-score of 0.264. This could be due some irregularities in dataset, but it is more likely a result of the design of the model. The model would thus still require substantial modifications to its architecture, or potentially the usage of a different kind of Neural Network to produce better results.

Lindqvist (2019) mentions having omitted the “fuzziness” feature from the original model by Schlüter and Böck (2014), as well as not linearly increasing the SGD optimiser’s momentum from 0.45 to 0.9 from the 10th to the 20th epoch. Perhaps implementing those missing features could help improve the performance of the model.

Anyhow, we hope that this project is a step in the right direction for the field of MIR, and that in the future, more experienced and specialised MIR researchers can take inspiration from this project, conduct research on Accented Onset Detection and find a better solution to the problem and produce better results.

6.2 Future Work

In the future, I would like to further work on the data pipeline, addressing the unequal distribution of characteristics across the dataset, by implementing the steps described in Section 3.5.4. Furthermore, I would like to extend the dataset by adding annotations for other types of articulations, such as staccatos, marcatos and tenutos. Once complete, I would like to make the dataset publicly available via a sharing platform such as GitHub or Kaggle.

I am also very eager on testing the idea of increasing the convolution window size, as mentioned in Section 5.2.1, and comparing its results with the Multilabel1 model.

References

Bello, J.P. and Sandler, M. (2003). Phase-based note onset detection for music signals. *IEEE Xplore*, [online] 5. doi:https://doi.org/10.1109/ICASSP.2003.1200001.

Böck, S., Schlüter, J. and Widmer, G. (2013). Enhanced peak picking for onset detection with recurrent neural networks.

Eyben, F., Böck, S., Schuller, B. and Graves, A. (2010). Univeral Onset Detection with Bidirectional Long Short-Term Memory Neural Networks. In: *11th International Society for Music Information Retrieval Conference (ISMIR 2010)*. pp.589–594.

Klapuri, A. (1999). Sound onset detection by applying psychoacoustic knowledge. In: *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No.99CH36258)*. [online] doi:https://doi.org/10.1109/icassp.1999.757494.

Lacoste, A. and Eck, D. (2005). Onset Detection with Artificial Neural Networks for MIREX 2005.

Lindqvist, B. (2019). Reproducing the State of the Art in Onset Detection Using Neural Networks. [online] Available at: https://www.diva-portal.org/smash/get/diva2:1338850/FULLTEXT02.pdf.

Marolt, M., Alenka Kavčič, M. Privošnik and Divjak, S. (2003). On detecting note onsets in piano music. doi:https://doi.org/10.1109/melecon.2002.1014600.

MuseScore.org. (n.d.). *Command line options*. [online] Available at: https://musescore.org/en/handbook/3/command-line-options

MuseScore.org. (n.d.). Silence at end of exported .mp3. [online] Available at: https://musescore.org/en/node/270923.

musicinformationretrieval.com. (n.d.). *musicinformationretrieval.com*. [online] Available at: https://musicinformationretrieval.com/.

nema.lis.illinois.edu. (2018). *MIREX 2018: Audio Onset Detection - MIREX05 Dataset - Introduction*. [online] Available at: https://nema.lis.illinois.edu/nema_out/mirex2018/results/aod/.

Pocaro, L. (n.d.). *Music Information Retrieval and its Application in Music Recommender Systems*. [online] algorithmic-transparency.ec.europa.eu. Available at: https://algorithmic-transparency.ec.europa.eu/events/music-information-retrieval-and-its-application-music-recommender-systems-2023-02-06_en.

Schedl, M., Gómez, E. and Urbano, J. (2014). Music Information Retrieval: Recent Developments and Applications. *Foundations and Trends® in Information Retrieval*, 8(2-3), pp.127–261. doi:https://doi.org/10.1561/1500000042.

Schlüter, J. and Böck, S. (2013). Musical Onset Detection with Convolutional Neural Networks.

Schlüter, J. and Böck, S. (2014). Improved musical onset detection with Convolutional Neural Networks. In: *International Conference on Acoustics, Speech, and Signal Processing*. doi:https://doi.org/10.1109/icassp.2014.6854953.

web.mit.edu. (n.d.). *music21 Documentation — music21 Documentation*. [online] Available at: https://web.mit.edu/music21/doc/index.html.

www.music-ir.org. (2005). *2005:Audio Onset Detection Results - MIREX Wiki*. [online] Available at: https://www.music-ir.org/mirex/wiki/2005:Audio_Onset_Detection_Results

Appendix A – musescore.com Scores Used for the Dataset

Name	Author	Source	URL	License
Aguas de Março (cover)	musescore.com user ElMaetrodeivi	musescore.com	https://musescore.com/user/18881881/scores/4057961	CC BY 4.0 DEED
Alright, O.K, You Win : Composed by Sid Wyche; Lyrics by Mayme Watts	musescore.com user fernpod	musescore.com	https://musescore.com/user/34378972/scores/8303453	CC BY 4.0 DEED
An Old Irish Blessing	musescore.com user eagri1967	musescore.com	https://musescore.com/user/8752261/scores/5040819	CC BY-NC-SA 4.0 DEED
Autumn Leaves for recorder quartet	musescore.com user Luis Ariel Delgado	musescore.com	https://musescore.com/user/19389076/scores/4659691	CC BY-NC 4.0 DEED
Bach - Chaconne from Violin Partita No.2 in d minor, BWV 1004	musescore.com user Eagle21	musescore.com	https://musescore.com/user/3017566/scores/5565651	CC BY 4.0 DEED
Bach Cello Suite 3 - Bourree I	musescore.com user saselberg	musescore.com	https://musescore.com/user/19193/scores/52585	PDM 1.0 DEED
Bass Clarinet Concerto K.622	musescore.com user SaXoPhone46093	musescore.com	https://musescore.com/user/8347661/scores/5705315	CC BY 4.0 DEED
Big Blue - Mario Kart 8	musescore.com user prearo	musescore.com	https://musescore.com/prearo/scores/5349273	CC BY-SA 4.0 DEED
Black and Blues	musescore.com user Shian-An Chiou	musescore.com	https://musescore.com/user/1174286/scores/5270217	CC BY-NC 4.0 DEED
Blue Line (Instrumental) - Gran Turismo 2	musescore.com user creet	musescore.com	https://musescore.com/creet/scores/6491707	CC BY-NC 4.0 DEED
Brahms – Intermezzo Op. 118 No. 2 (A Major) [String Quartet Arrangement]	musescore.com user Leonardo Cisija	musescore.com	https://musescore.com/nardo/scores/4083066	CC BY-NC-SA 4.0 DEED

Caro Mio Ben clarinetti fagotto glockenspiel	musescore.com user danisav	musescore.com	https://musescore.com/user/5373351/scores/9839341	CC BY-NC-SA 4.0 DEED
Coldplay - Viva la Vida (for String Orchestra)	musescore.com user oreboy	musescore.com	https://musescore.com/oreboy/scores/4177661	PDM 1.0 DEED
Count Bubba (Lead Trumpet Transcription)	musescore.com user That One Jazz Kid	musescore.com	https://musescore.com/user/30573755/scores/6078642	CC BY-NC-SA 4.0 DEED
Daddy Lessons Rough	musescore.com user elsalopez1029	musescore.com	https://musescore.com/user/33222395/scores/5769798	CC BY 4.0 DEED
Daft Punk- Something About Us	musescore.com user Cristiana Simoes	musescore.com	https://musescore.com/user/9405616/scores/5353450	CC BY 4.0 DEED
dark fantasy jazz - Ariel Ruzitsky	musescore.com user ariruz25	musescore.com	https://musescore.com/user/35673490/scores/13411903	CC BY 4.0 DEED
Daughter of Hallownest - Violin Solo	musescore.com user 5Gonza5	musescore.com	https://musescore.com/user/26033606/scores/6997774	CC BY-SA 4.0 DEED
Deadlocked	musescore.com user ruron2	musescore.com	https://musescore.com/user/19994781/scores/3899386	CC BY-NC 4.0 DEED
Don't Stop Skankin'	musescore.com user Jack_Sneddon	musescore.com	https://musescore.com/user/59802/scores/1375136	CC BY 4.0 DEED
Drake - Passionfruit	musescore.com user paramike97	musescore.com	https://musescore.com/user/13465726/scores/5792311	CC BY 4.0 DEED
Eastern Steam Dream Quartet	musescore.com user Khezk	musescore.com	https://musescore.com/user/33306288/scores/6312165	CC BY-NC-SA 4.0 DEED
Emmanuel – Michel Colombier Michel Colombier - Emmanuel	musescore.com user cbenassaya	musescore.com	https://musescore.com/user/591833/scores/11021104	CC BY-NC 4.0 DEED
España Cañi	musescore.com user av8rpat	musescore.com	https://musescore.com/user/9481/scores/5067050	PDM 1.0 DEED
Fandango de Lleitariegos	musescore.com user Victor A. gaita	musescore.com	https://musescore.com/user/3174	CC BY 4.0 DEED

			4048/scores/7011926	
Fine and Dandy – Leon Donaldson - 1904	musescore.com user pwk	musescore.com	https://musescore.com/user/73894/scores/7859435	CC BY 4.0 DEED
Fly Me To The Moon Sax Quartet	musescore.com user Chase_of_Spades	musescore.com	https://musescore.com/user/18055971/scores/5233300	CC BY-NC 4.0 DEED
For Lorraine - by David Petersen	musescore.com user drdavidpeters	musescore.com	https://musescore.com/user/66764458/scores/12820288	CC BY-NC-ND 4.0 DEED
Ghost Town (Rico's Solo)	musescore.com user MarcGuer	musescore.com	https://musescore.com/user/1116011/scores/6148011	CC BY 4.0 DEED
Gigue for Horn and Organ from the First Cello Suite - J.S. Bach/Spielmann	musescore.com user HornSpiel	musescore.com	https://musescore.com/user/55575753/scores/8959144	CC BY-NC-SA 4.0 DEED
Gymnopédie 1	musescore.com user pcarmich	musescore.com	https://musescore.com/user/1190/scores/29029	CC BY-SA 4.0 DEED
Händel - Lascia Ch'io Pianga, for Flute and Harp	musescore.com user Hyeon Kim	musescore.com	https://musescore.com/user/16112556/scores/3390751	CC BY-SA 4.0 DEED
Happy Together (The Turtles) - quinteto	musescore.com user Olalli	musescore.com	https://musescore.com/user/27713608/scores/5727821	CC BY 4.0 DEED
How Deep Is Your Love Bari Sax/Bass Clarinet Duet	musescore.com user clubbedsam	musescore.com	https://musescore.com/user/31526849/scores/5792075	CC BY 4.0 DEED
I Am Easy - Flute, Guitar, Cello, Harmonica	musescore.com user Julio Silveira	musescore.com	https://musescore.com/user/11179056/scores/6403527	CC BY 4.0 DEED
I do, I do, I do, I do, I do - B Andersson & B Ulvaeus (ABBA) Brass Quintet	musescore.com user Peet du Toit	musescore.com	https://musescore.com/user/19658656/scores/4225251	CC BY-ND 4.0 DEED
I Talk To The Wind - (King Crimson)	musescore.com user ZolloKaptain	musescore.com	https://musescore.com/user/4011246/scores/5792125	CC BY 4.0 DEED

It's Raining Somewhere Else	musescore.com user knightraven888	musescore.com	https://musescore.com/user/17668896/scores/5709696	CC BY 4.0 DEED
J.S. Bach - Magnificat "Deposuit..." for Sax tenor & Keyboard	musescore.com user Aquarius 2.02	musescore.com	https://musescore.com/user/8108371/scores/5570686	CC BY-SA 4.0 DEED
Jesus Christ you are my life - M. Frisina	musescore.com user steve68	musescore.com	https://musescore.com/user/9057676/scores/2020866	CC BY-SA 4.0 DEED
John Dowland: Flow my tears	musescore.com user H. Gürtler	musescore.com	https://musescore.com/user/10806561/scores/6039121	CC BY-NC-SA 4.0 DEED
Jungfräulein, soll ich mit euch gehn - Horn in F and Marimba Duet	musescore.com user DoubleA/24	musescore.com	https://musescore.com/user/30313799/scores/6599678	CC BY 4.0 DEED
K.5 Flowers, Leaves, and Rain Clouds - Mikazuki 三日月	musescore.com user mikazukiworks	musescore.com	https://musescore.com/user/30554090/scores/7977356	CC BY-NC-ND 4.0 DEED
Latin Love Brass Medley (Brass Quintet)	musescore.com user Peet du Toit	musescore.com	https://musescore.com/user/19658656/scores/5609442	CC BY 4.0 DEED
Lea! - CrossCode	musescore.com user GMA SheetMusic	musescore.com	https://musescore.com/gma_sheetmusic/scores/5821683	CC BY-SA 4.0 DEED
Leave The Door Open (Lead Sheet Style)	musescore.com user JJ The Omega	musescore.com	https://musescore.com/user/27594453/scores/6689391	CC BY 4.0 DEED
Libertango - Contrabaixo e Piano	musescore.com user m.2002.p	musescore.com	https://musescore.com/user/29555794/scores/5630494	CC BY-NC 4.0 DEED
Manhattan Beach March Trumpet Trio	musescore.com user stepienalex	musescore.com	https://musescore.com/user/34129195/scores/5188351	CC BY 4.0 DEED
Marche Comique Revised	musescore.com user PanamaPiLover	musescore.com	https://musescore.com/user/326746/scores/612186	PDM 1.0 DEED
Maria Lisboa	musescore.com user Do-it	musescore.com	https://musescore.com/user/4276146/scores/1783496	PDM 1.0 DEED

math rock I guess	musescore.com user londonsharar	musescore.com	https://musescore.com/user/44989922/scores/8020020	CC0 1.0 DEED
Měsíčku na nebi hlubokém	musescore.com user capoverde	musescore.com	https://musescore.com/capoverde/scores/5252499	CC BY-NC-ND 4.0 DEED
Metallica - Nothing Else Matters	musescore.com user Lemanuel DT	musescore.com	https://musescore.com/user/9622861/scores/4414946	CC BY-SA 4.0 DEED
Milky Way (Explore)	musescore.com user sharp_shooter	musescore.com	https://musescore.com/user/4776246/scores/1991056	CC BY 4.0 DEED
Moon River - Andy Williams - Woodwind Quintet	musescore.com user saritaasulk	musescore.com	https://musescore.com/user/34300810/scores/6067107	CC BY 4.0 DEED
Northern Lights for Brass Trio	musescore.com user Lbaeza123	musescore.com	https://musescore.com/user/13466166/scores/4446956	CC BY 4.0 DEED
Pavane de Fauré pour hautbois et violon	musescore.com user Goldenaie	musescore.com	https://musescore.com/user/14781256/scores/5740942	CC BY-ND 4.0 DEED
Rumba de Laroá (Pel+Madeira.I.24)	musescore.com user Johan Van der Elst	musescore.com	https://musescore.com/user/69096/scores/10124041	CC BY-NC-SA 4.0 DEED
Sad Song - We The Kings (a Willy Deals Arrangement)	musescore.com user Willy Deals	musescore.com	https://musescore.com/user/26056/scores/4793546	CC BY 4.0 DEED
Scotland The Brave – Misc tunes Scotland The Brave : "Scotland the Brave, ou les Aventures en Ecosse de six thons"	musescore.com user Jad-Alex Corzoma Seif	musescore.com	https://musescore.com/user/27479354/scores/8912928	CC BY 4.0 DEED
Shadows The melodia solo-Apache 2 - GUITARE SOLO - 2021-01-14 1200	musescore.com user Mozart24111	musescore.com	https://musescore.com/user/13574/scores/6553367	CC BY 4.0 DEED
Sky Is Over- Serj Tankian	musescore.com user Instrumental Rock	musescore.com	https://musescore.com/ir/scores/5088980	CC BY 4.0 DEED

Sonata B Flat - Major for Bassoon and Piano - Jerome Besozzi	musescore.com user Georg Pfeifer	musescore.com	https://musescore.com/user/37957/scores/6646576	CC BY-NC-SA 4.0 DEED
Sonata Nr. 3	musescore.com user veronicabafol1	musescore.com	https://musescore.com/user/30975446/scores/5693093	CC BY-NC 4.0 DEED
Super Mario 3D World Main Theme	musescore.com user thepokemonfan	musescore.com	https://musescore.com/user/75656/scores/174220	CC BY-NC-SA 4.0 DEED
Superstition – Stevie Wonder (For jazz sextet)	musescore.com user Roberto perkinos	musescore.com	https://musescore.com/user/59041438/scores/10853506	CC BY 4.0 DEED
Swan Lake Theme	musescore.com user itsnotokay	musescore.com	https://musescore.com/user/3559611/scores/4833870	CC BY-SA 4.0 DEED
Sweet Child O' Mine - Guns N' Roses	musescore.com user Vader915	musescore.com	https://musescore.com/user/9106196/scores/5694331	CC BY 4.0 DEED
System of a Down - Radio/Video	musescore.com user lukiduk	musescore.com	https://musescore.com/user/157684/scores/2427156	CC BY-NC-SA 4.0 DEED
TANGO: POR UNA CABEZA	musescore.com user thedawnofcrystals	musescore.com	https://musescore.com/user/25809721/scores/5604401	CC BY 4.0 DEED
Telemann, G. P. _ Trio Sonata f-moll, TWV 42:f2 [Rec, Ob, Bsn]	musescore.com user hachi.ike	musescore.com	https://musescore.com/user/3795306/scores/6326064	CC BY 4.0 DEED
The Dancing Bull (A Duet for Trombone and French Horn)	musescore.com user HoneyBadger159	musescore.com	https://musescore.com/user/9224571/scores/2040026	CC BY 4.0 DEED
The Kings Wish (Solo Vibes)	musescore.com user iike_	musescore.com	https://musescore.com/user/27604158/scores/5776129	CC BY 4.0 DEED
The Way We Were (Xylophone & 2 Marimba Arr.)	musescore.com user Person of Hourai	musescore.com	https://musescore.com/user/45956/scores/5682952	CC BY 4.0 DEED
Two Butterflies Duet for Oboe and Bassoon in D Major	musescore.com user vsr83	musescore.com	https://musescore.com/user/29878806/scores/5971998	CC BY-NC 4.0 DEED

Verdi - La Traviata: Parigi o cara, for Flute and Cello	musescore.com user Hyeon Kim	musescore.com	https://musescore.com/user/16112556/scores/3594666	CC BY-SA 4.0 DEED
Victory Fanfare	musescore.com user maevelander	musescore.com	https://musescore.com/maevelander/scores/5916770	CC BY 4.0 DEED
Walking bassline	musescore.com user MusicLover419	musescore.com	https://musescore.com/user/30914341/scores/6271177	CC BY 4.0 DEED
Walking in the Village	musescore.com user babadem19	musescore.com	https://musescore.com/user/32888226/scores/5865402	CC BY-NC-SA 4.0 DEED
What an odd duet (100 note challenge)	musescore.com user nfaist	musescore.com	https://musescore.com/user/183088/scores/978751	CC BY-NC-SA 4.0 DEED
Waltz No. 28	musescore.com user Victor Sinadinoski	musescore.com	https://musescore.com/user/46167530/scores/7913921	CC BY-NC-ND 4.0 DEED
You Say Run - Violin Solo (Main Part Only)	musescore.com user graemestocker	musescore.com	https://musescore.com/user/32930580/scores/5716436	CC BY 4.0 DEED
Young Dragon's Dance	musescore.com user Freb	musescore.com	https://musescore.com/freb/scores/6628739	CC BY-SA 4.0 DEED
То не ветер ветку клонит (It's not the Wind that bends the Branch)	musescore.com user Serge Lagoutine	musescore.com	https://musescore.com/user/2394341/scores/831561	CC BY-SA 4.0 DEED

Appendix B – Risk Assessment

Description of Risk	Impact of Risk	Likelihood Rating	Impact Rating	Preventative Actions
Low Motivation	Leads to unproductive mindset and low effort put into the project.	Low/Medium	Medium/High	Break my tasks down. Do not hesitate to ask for guidance from my supervisor.
Bad Organisation	Leads overwork / underwork, depending on my priorities.	Medium/High	Medium/High	Organise my weeks, make sure I keep a healthy balance between work and external activities.
Writing bad, uncommented and unorganised code	Leads to project not working, too much time wasted on debugging, project may not work by the deadline	Low/Medium	High	Document every bit of progress I make after each coding session.
Loss of progress	Important code is deleted, I may not be able to recall what I wrote or what exact approaches I took.	Low	High	Regularly commit my changes to my GitHub repository, to make sure that everything I do is saved.
Laptop crashes / lost / destroyed	Recent unsaved progress will be lost.	Medium	Low/Medium	Only unsaved progress will be lost, as the rest is saved in my GitHub repository.
Insufficient technical capabilities	Leads to blockage and writing code without knowing what I'm doing, resulting in unproductive working sessions.	Medium	Medium/High	Conduct sufficient research and preparation in order to expand my knowledge on the relevant subjects.